Équations Linéaires

par Léo Peyronnet

Octobre 2022

Compte rendu du TP consistant à programmer et comparer certaines méthodes de résolution de systèmes linéaires.

1 Rappel des méthodes

1.1 Méthode de Gauss (directe)

Dans le système $Ax = b$, la matrice inversible $A \in M_{n,n}(\mathbb{R})$ s'exprime sous la forme $A = M.N$ avec les matrices M et N où M est facilement inversible et N est triangulaire. Ainsi, $Ax = b \Leftrightarrow M.Nx = b \Leftrightarrow Nx = M^{-1}b$. La méthode de Gauss consiste en deux étapes :

— La triangulation : où l'on cherche $y \in \mathbb{R}^n$ tel que $My = b \Leftrightarrow y = M^{-1}b$.

— La résolution "facile" : où l'on cherche $x \in \mathbb{R}^n$ tel que $Nx = b \Leftrightarrow x = N^{-1}y$.

1.2 Méthodes itératives

Toujours dans le système $Ax = b$, la matrice inversible $A \in M_{n,n}(\mathbb{R})$ peut également s'exprimer sous la forme $A = M - N$ avec M facilement inversible. Ainsi, $Ax = b \Leftrightarrow (M - N)x = b \Leftrightarrow Mx - Nx = b \Leftrightarrow Mx = Nx + b \Leftrightarrow x = M^{-1}(Nx + b)$.

Soit $F(x) = M^{-1}(Nx+b)$ tel que $F(x) = x$ pour la solution de $Ax = b$. x est donc un point fixe de la fonction $F(x)$. Si $F(x)$ est une application strictement contractante, alors la suite ci-dessous converge vers un point fixe de F(x).

$$\begin{cases} x_0 \in \mathbb{R}^n \\ x_{n+1} = F(x_n) \end{cases}$$

$$\Leftrightarrow \begin{cases} x_0 \in \mathbb{R}^n \\ x_{n+1} = M^{-1}(Nx_n + b) \end{cases}$$

1.2.1 Méthode Jacobi

Décomposition: $A = D - E - F$ avec :

— D la diagonale de A.

— $-E$ la partie sous la diagonale de A.

— $-F$ la partie sur la diagonale de A.

Ainsi, $M = D$; $N = E + F$; $A = M - N$.

Récurrence:

$$\begin{cases} x_0 \in \mathbb{R}^n \\ x_{n+1} = D^{-1}((E+F)x_n + b)) \end{cases}$$

Convergence: Si la matrice A a une diagonale strictement dominante, alors $F(x) = M^{-1}(Nx + b)$ est strictement contractante et x_n tend vers la solution.

1.2.2 Méthode Gauss-Seidel

Décomposition: Avec la même décomposition,
$M = D - E$; $N = F$; $A = M - N$.

Récurrence:

$$\begin{cases} x_0 \in \mathbb{R}^n \\ x_{n+1} = (D-E)^{-1}(Fx_n + b)) \end{cases}$$

Convergence: Si la matrice A a une diagonale strictement dominante ou si A est symétrique définie positive, alors $F(x) = M^{-1}(Nx + b)$ est strictement contractante et x_n tend vers la solution.

2 Présentation des programmes

2.1 Algorithme de Gauss simple et résolution de matrice triangulaire supérieure

```
void gauss(float ** A,float * B,int taille){
    for (int k=0;k<taille-1;k++){
        for (int i=k+1;i<taille;i++){
            float piv=A[i][k]/A[k][k];
            for (int j=k;j<taille;j++){
                A[i][j]-=piv*A[k][j];
            }
            B[i]-=piv*B[k];
        }
    }
}

float * ResTrigSup(float ** A,float * B,int taille){
    float * X=declTab(taille);
    int n=taille-1;

    X[n]=B[n]/A[n][n];
    for (int i=n-1;i>=0;i--){
        float somm=0;
        for (int j=i+1;j<=n;j++){
            somm+=A[i][j]*X[j];
        }
        X[i]=(1/A[i][i])*(B[i]-somm);
    }
    return X;
```

```
}
```

2.2 Méthode Jacobi

```
float E(float ** A,float * X,float * B,int taille){
    float norme=0;
    float v=0;
    for (int i=0;i<taille;i++){
        //Produit Vectoriel A*X
        float AXi=0; //Somme du produit A[i]*X[i]
        for (int j=0;j<taille;j++){
            AXi+=A[i][j]*X[i];
        }
        v=AXi-B[i]; // v représente le produit vectoriel AX de la i-ème ligne auquel on soustrait B[i]
        norme+=puiss(v,2); //somme des v de toutes les lignes ("taille" lignes) au carré
    }
    return sqrtf(norme); // <=> ||AX-B||
}

float * jacobi(float e,float ** A,float * B,int taille){
    //initialisation de X (x0)
    float * X=declTab(taille);
    for (int y=0;y<taille;y++){
        X[y]=1/A[y][y]*B[y]; //première estimation de X
    }
    //algo jacobi
    //RAPPEL: Ax=b <=> A=D-E-F
    int k=1;
    while (E(A,X,B,taille)>=(1/e)){
        for (int i=0;i<taille;i++){
            //Produit Vectoriel -((E+F)*x) pour la i-ème ligne
            float somm=0;
            for (int j=0;j<taille;j++){
                if (j!=i){
                    somm+=A[i][j]*X[j];
                }
            }
            //
            X[i]=((1/A[i][i])*(B[i]-somm)); // <=> D-1 * (B-(-(E+F)*x))
        }
        k++;
    }
    printf("k=%d\n",k); //affichage du nombre d'itérations
    return X;
}
```

2.3 Méthode Gauss-Seidel

```
// La fonction E est sous-entendue

float * gauss_seidel(float e,float ** A,float * B,int taille){
```

```
    //initialisation de X (x1)
    float * X0=declTab(taille);
    float * X1=declTab(taille);
    for (int y=0;y<taille;y++){
        X1[y]=1/A[y][y]*B[y]; //première estimation de X
    }
    //algo gauss_seidel
    int k=1;
    while (E(A,X1,B,taille)>=(1/e)){
        for (int i=0;i<taille;i++){
            float somm=0;
            for (int j=0;j<i;j++){
                somm+=A[i][j]*X1[j];
            }
            for (int j=i+1;j<taille;j++){
                somm+=A[i][j]*X0[j];
            }
            X1[i]=((1/A[i][i])*(B[i]-somm));
            X0[i]=X1[i];
        }
        k++;
    }
    printf("k=%d\n",k); //affichage du nombre d'itérations
    return X1;
}
```

3 Tables et graphiques sur les jeux d'essais

4 Conclusion